ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Multi-level Trace Visualization

Presented by: Naser Ezzati Jivan
Supervisor: Professor Michel Dagenais

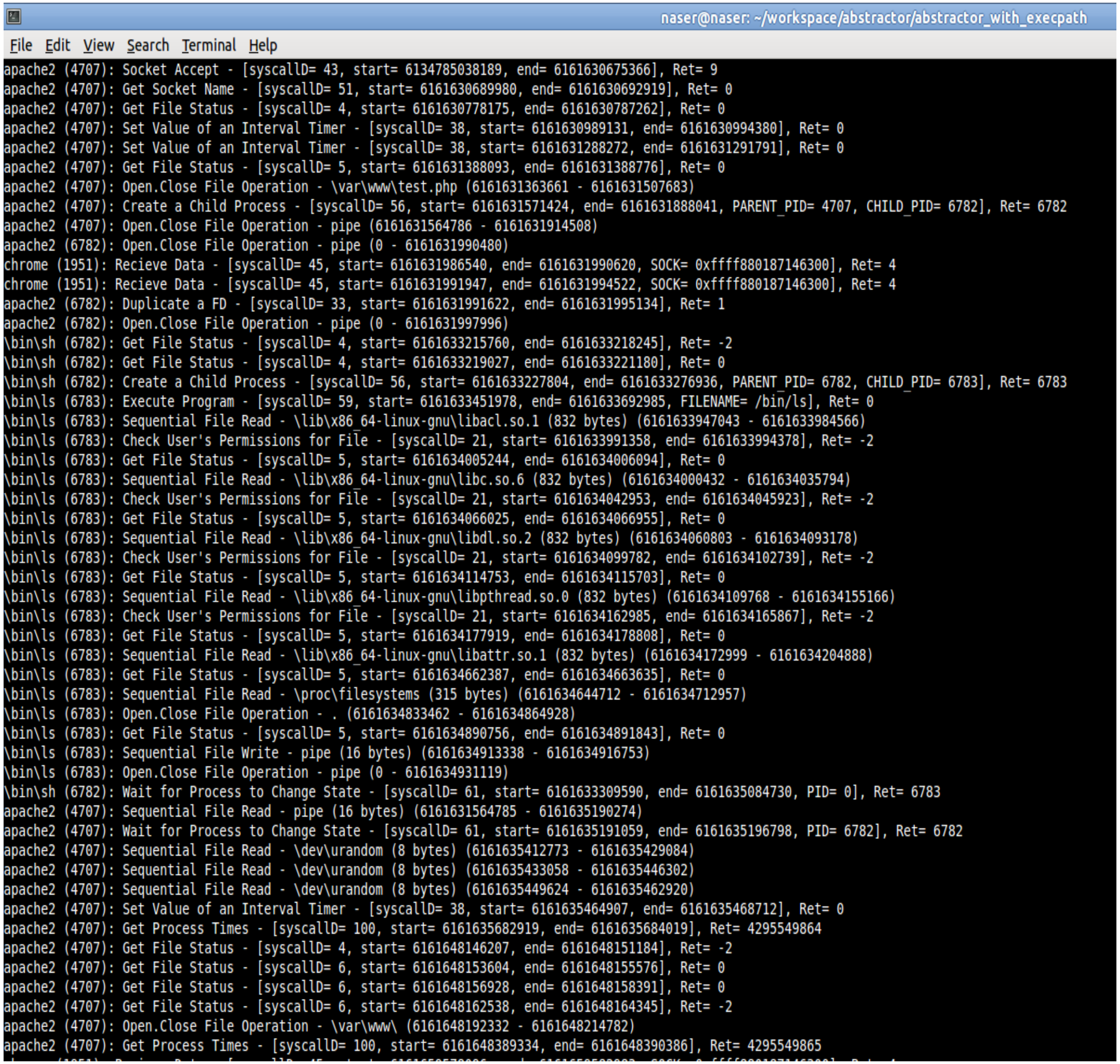
DORSAL LAB

École Polytechnique, Montreal

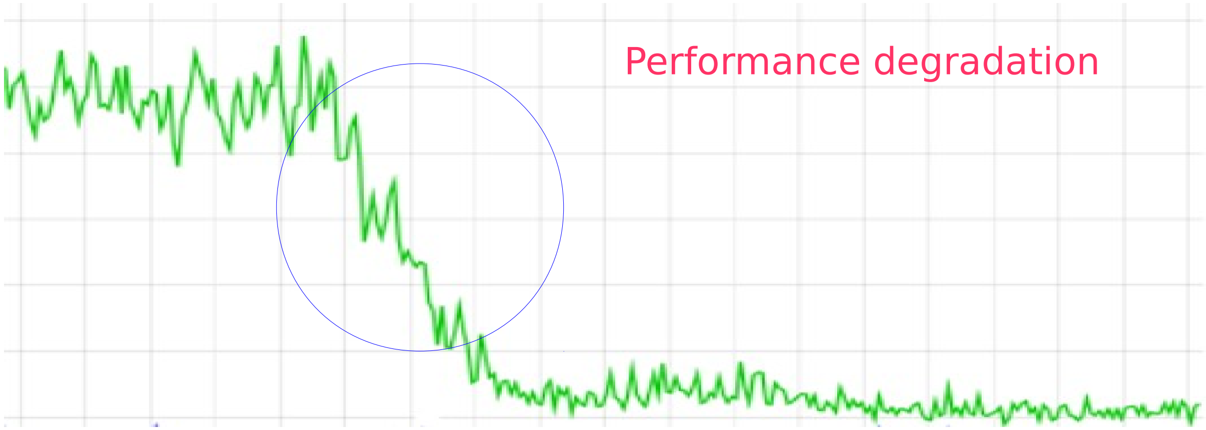
Overview

- Data store
  - Linking the different layers
- Hierarchical visualization techniques
  - LoD-based control flow diagram
    - Label placement
  - “Node-link + Treemap” visualization

How to explore and understand ?

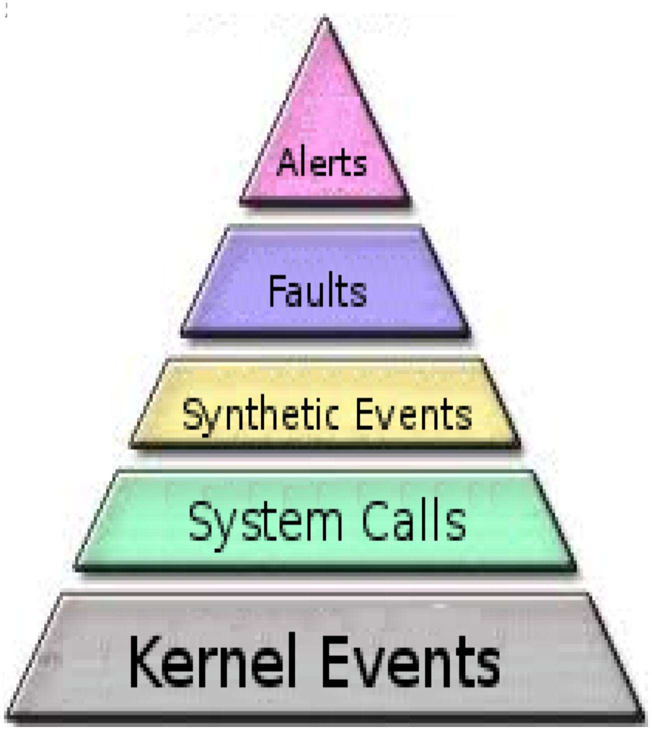


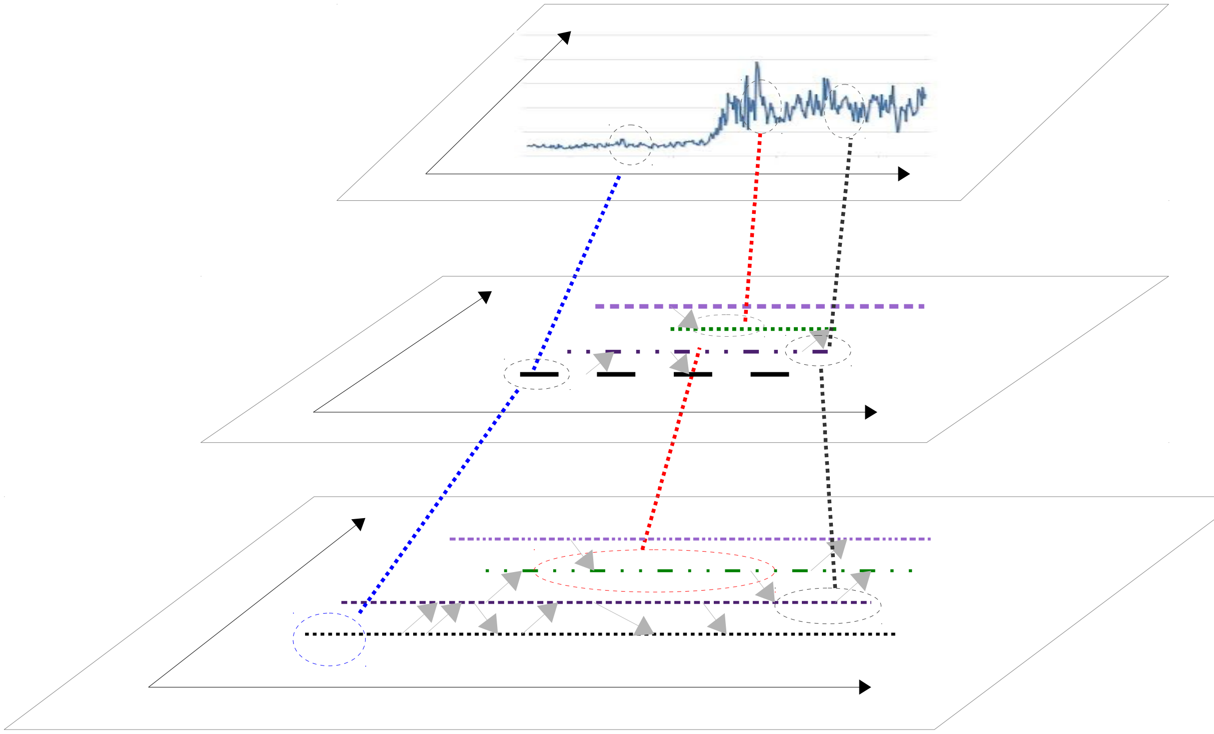
```
naser@naser: ~/workspace/abstractor/abstractor_with_execpath
File Edit View Search Terminal Help
apache2 (4707): Socket Accept - [syscallD= 43, start= 6134785038189, end= 6161630675366], Ret= 9
apache2 (4707): Get Socket Name - [syscallD= 51, start= 6161630689980, end= 6161630692919], Ret= 0
apache2 (4707): Get File Status - [syscallD= 4, start= 6161630778175, end= 6161630787262], Ret= 0
apache2 (4707): Set Value of an Interval Timer - [syscallD= 38, start= 6161630989131, end= 6161630994380], Ret= 0
apache2 (4707): Set Value of an Interval Timer - [syscallD= 38, start= 6161631288272, end= 6161631291791], Ret= 0
apache2 (4707): Get File Status - [syscallD= 5, start= 6161631388093, end= 6161631388776], Ret= 0
apache2 (4707): Open.Close File Operation - \var\www\test.php (6161631363661 - 6161631507683)
apache2 (4707): Create a Child Process - [syscallD= 56, start= 6161631571424, end= 6161631888041, PARENT_PID= 4707, CHILD_PID= 6782], Ret= 6782
apache2 (4707): Open.Close File Operation - pipe (6161631564786 - 6161631914508)
apache2 (6782): Open.Close File Operation - pipe (0 - 6161631990480)
chrome (1951): Recieve Data - [syscallD= 45, start= 6161631986540, end= 6161631990620, SOCK= 0xffff880187146300], Ret= 4
chrome (1951): Recieve Data - [syscallD= 45, start= 6161631991947, end= 6161631994522, SOCK= 0xffff880187146300], Ret= 4
apache2 (6782): Duplicate a FD - [syscallD= 33, start= 6161631991622, end= 6161631995134], Ret= 1
apache2 (6782): Open.Close File Operation - pipe (0 - 6161631997996)
\bin\sh (6782): Get File Status - [syscallD= 4, start= 6161633215760, end= 6161633218245], Ret= -2
\bin\sh (6782): Get File Status - [syscallD= 4, start= 6161633219027, end= 6161633221180], Ret= 0
\bin\sh (6782): Create a Child Process - [syscallD= 56, start= 6161633227804, end= 6161633276936, PARENT_PID= 6782, CHILD_PID= 6783], Ret= 6783
\bin\ls (6783): Execute Program - [syscallD= 59, start= 6161633451978, end= 6161633692985, FILENAME= /bin/ls], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libacl.so.1 (832 bytes) (6161633947043 - 6161633984566)
\bin\ls (6783): Check User's Permissions for File - [syscallD= 21, start= 6161633991358, end= 6161633994378], Ret= -2
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634005244, end= 6161634006094], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libc.so.6 (832 bytes) (6161634000432 - 6161634035794)
\bin\ls (6783): Check User's Permissions for File - [syscallD= 21, start= 6161634042953, end= 6161634045923], Ret= -2
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634066025, end= 6161634066955], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libdl.so.2 (832 bytes) (6161634060803 - 6161634093178)
\bin\ls (6783): Check User's Permissions for File - [syscallD= 21, start= 6161634099782, end= 6161634102739], Ret= -2
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634114753, end= 6161634115703], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libpthread.so.0 (832 bytes) (6161634109768 - 6161634155166)
\bin\ls (6783): Check User's Permissions for File - [syscallD= 21, start= 6161634162985, end= 6161634165867], Ret= -2
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634177919, end= 6161634178808], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libattr.so.1 (832 bytes) (6161634172999 - 6161634204888)
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634662387, end= 6161634663635], Ret= 0
\bin\ls (6783): Sequential File Read - \proc\filesystems (315 bytes) (6161634644712 - 6161634712957)
\bin\ls (6783): Open.Close File Operation - . (6161634833462 - 6161634864928)
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634890756, end= 6161634891843], Ret= 0
\bin\ls (6783): Sequential File Write - pipe (16 bytes) (6161634913338 - 6161634916753)
\bin\ls (6783): Open.Close File Operation - pipe (0 - 6161634931119)
\bin\sh (6782): Wait for Process to Change State - [syscallD= 61, start= 6161633309590, end= 6161635084730, PID= 0], Ret= 6783
apache2 (4707): Sequential File Read - pipe (16 bytes) (6161631564785 - 6161635190274)
apache2 (4707): Wait for Process to Change State - [syscallD= 61, start= 6161635191059, end= 6161635196798, PID= 6782], Ret= 6782
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635412773 - 6161635429084)
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635433058 - 6161635446302)
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635449624 - 6161635462920)
apache2 (4707): Set Value of an Interval Timer - [syscallD= 38, start= 6161635464907, end= 6161635468712], Ret= 0
apache2 (4707): Get Process Times - [syscallD= 100, start= 6161635682919, end= 6161635684019], Ret= 4295549864
apache2 (4707): Get File Status - [syscallD= 4, start= 6161648146207, end= 6161648151184], Ret= -2
apache2 (4707): Get File Status - [syscallD= 6, start= 6161648153604, end= 6161648155576], Ret= 0
apache2 (4707): Get File Status - [syscallD= 6, start= 6161648156928, end= 6161648158391], Ret= 0
apache2 (4707): Get File Status - [syscallD= 6, start= 6161648162538, end= 6161648164345], Ret= -2
apache2 (4707): Open.Close File Operation - \var\www\ (6161648192332 - 6161648214782)
apache2 (4707): Get Process Times - [syscallD= 100, start= 6161648389334, end= 6161648390386], Ret= 4295549865
```




```
\bin\ls (6783): Open.Close File Operation - . (6161634833462 - 6161634864928)
\bin\ls (6783): Get File Status - [syscallD= 5, start= 6161634890756, end= 6161634891843], Ret= 0
\bin\ls (6783): Sequential File Write - pipe (16 bytes) (6161634913338 - 6161634916753)
\bin\ls (6783): Open.Close File Operation - pipe (0 - 6161634931119)
\bin\sh (6782): Wait for Process to Change State - [syscallD= 61, start= 6161633309590, end= 6161635084730, PID= 0], Ret= 6783
```

Unified View





How to model and visualize?

Challenges?

Statistics Cubes

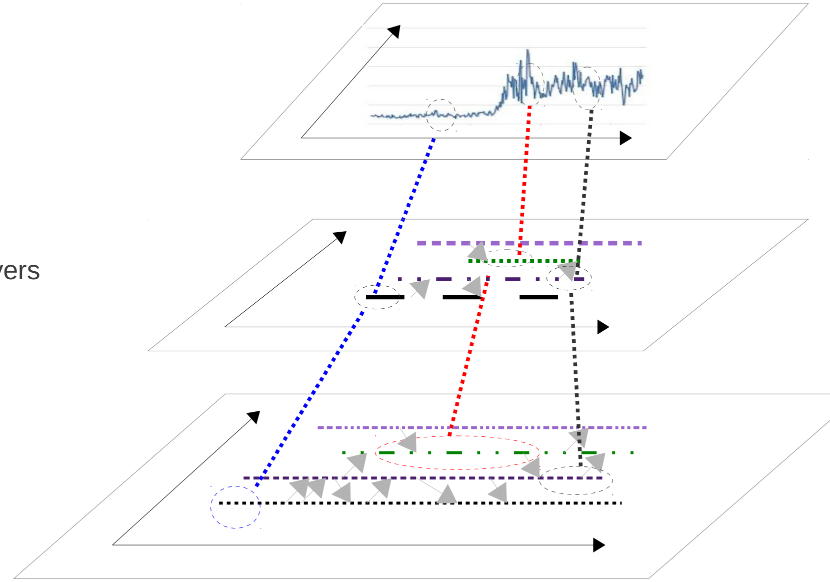
Faults and Alerts

Synthetic Events

Semantic Events

Linking different layers

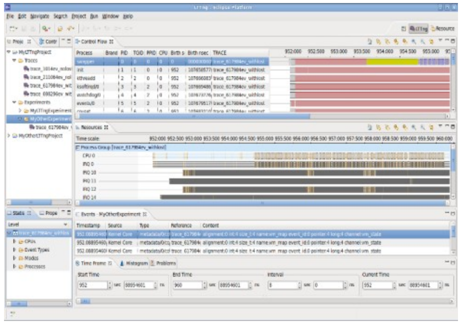
2



Trace abstraction &
fault Identification

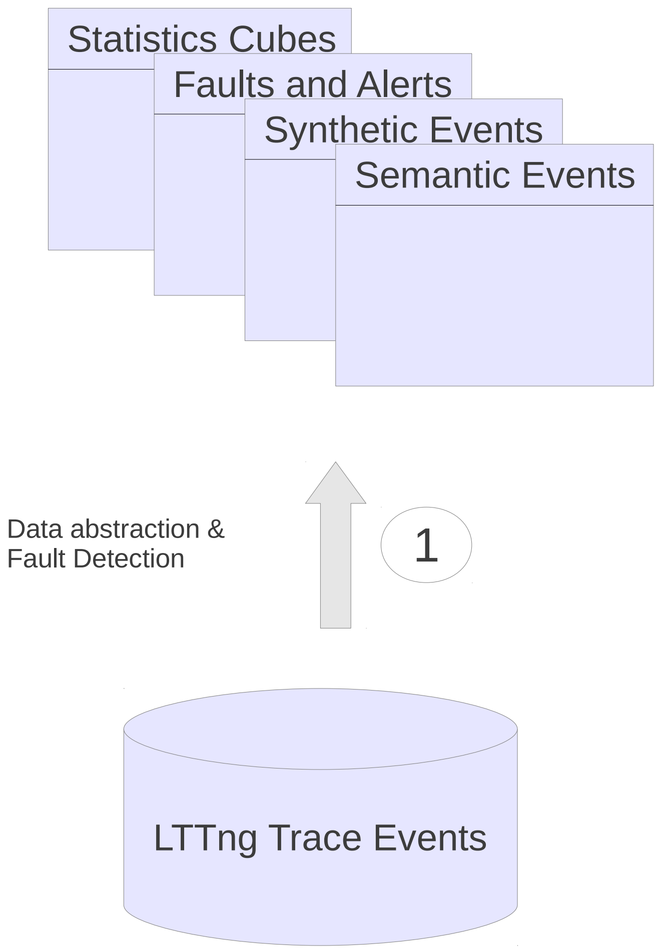
1

3



LTTng Trace Events

1 How to create different levels of information?

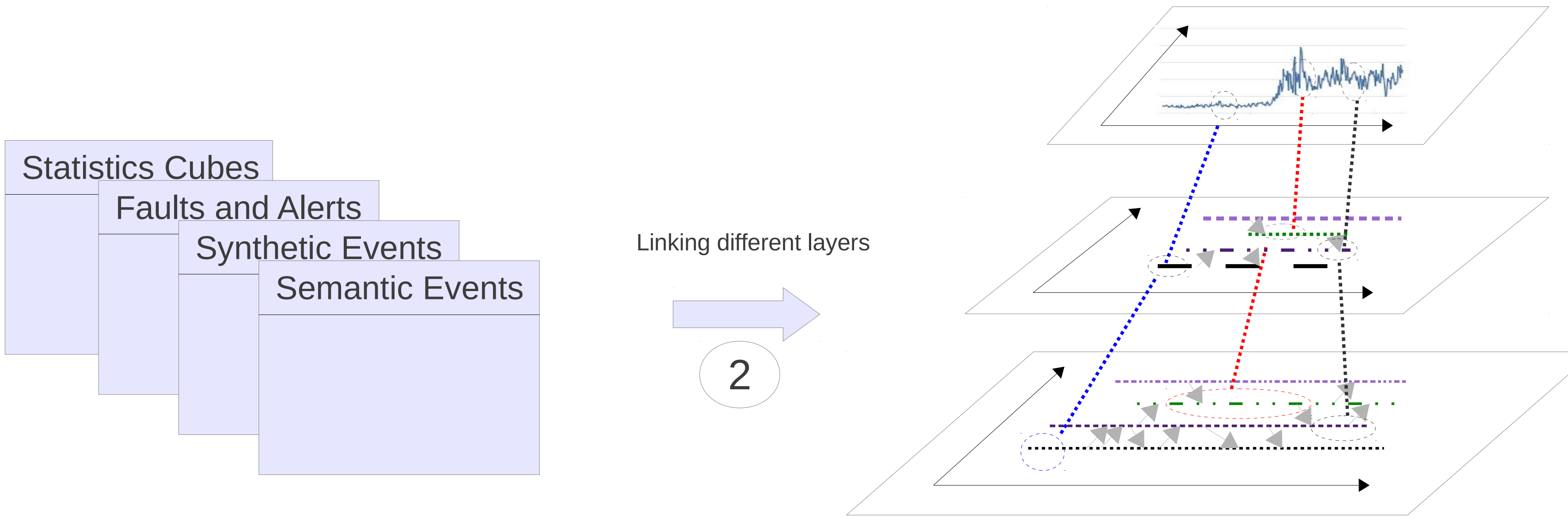


Trace Abstraction & Fault Detection

- Pattern matching
- Frequent pattern mining
- Learning

  - Concordia
  - Laval University

2 How to link the different levels?

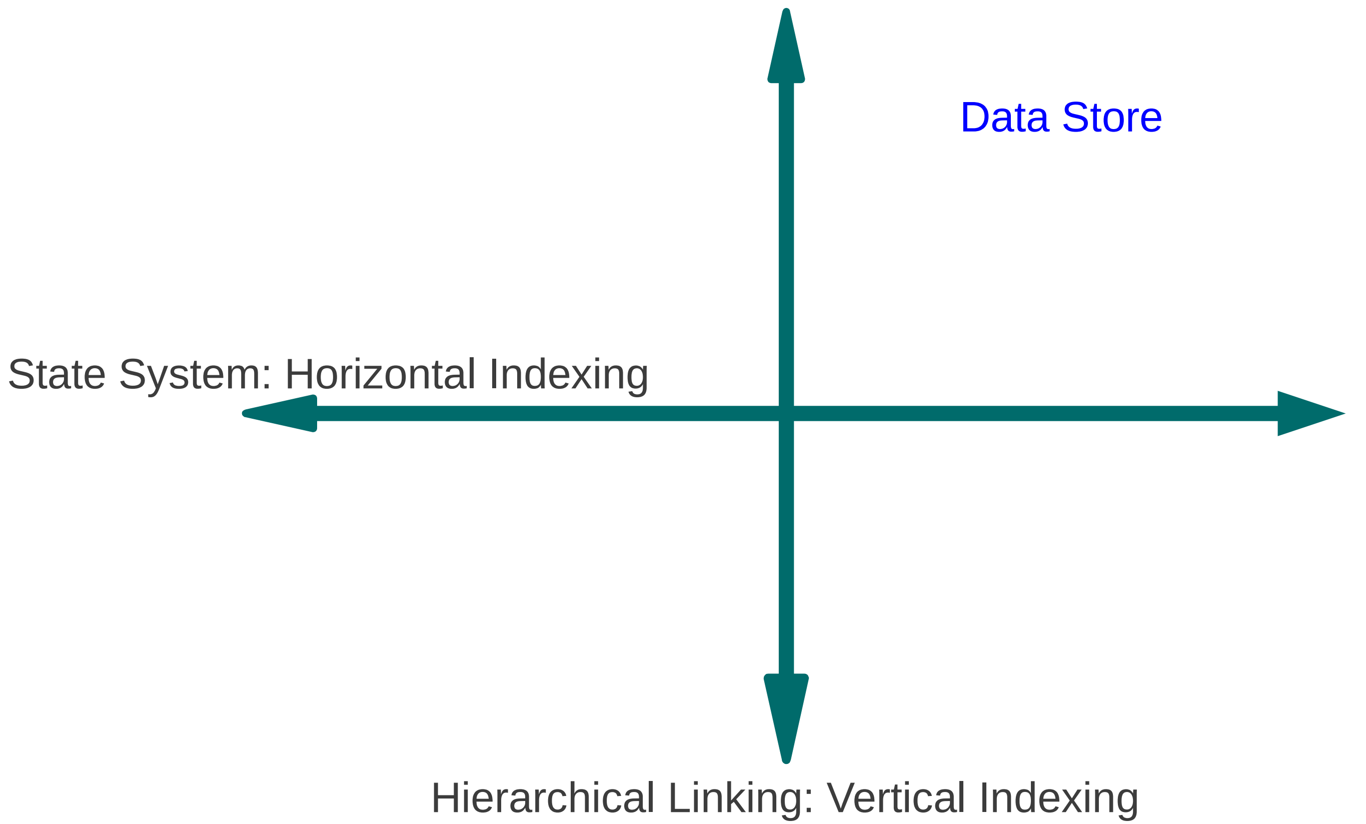


Relating the different levels enables:

- A multi-resolution analysis of the system under study and a better comprehension.
- Following and digging into a detected problem to find more details.

2

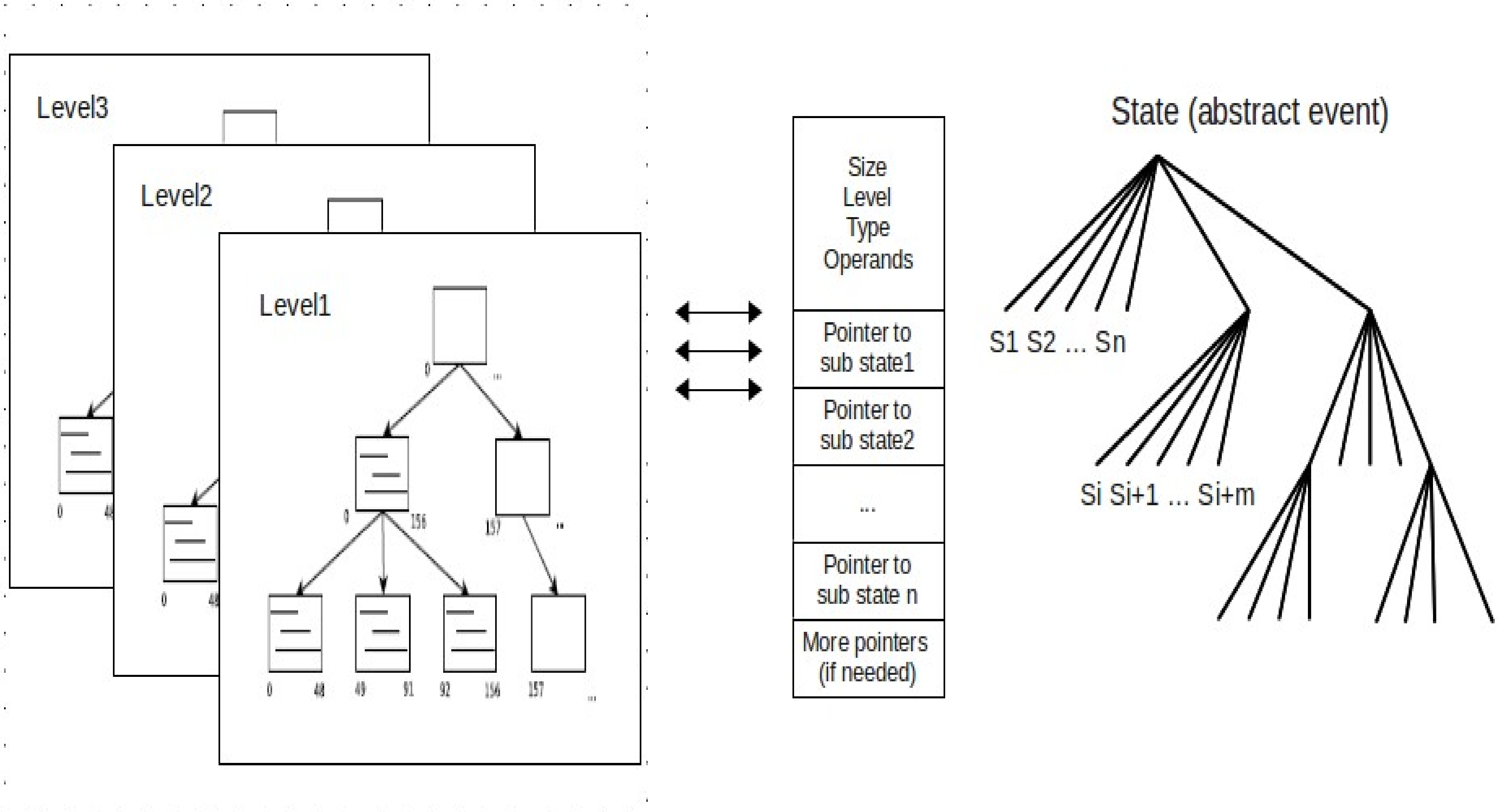
How to link the different levels?



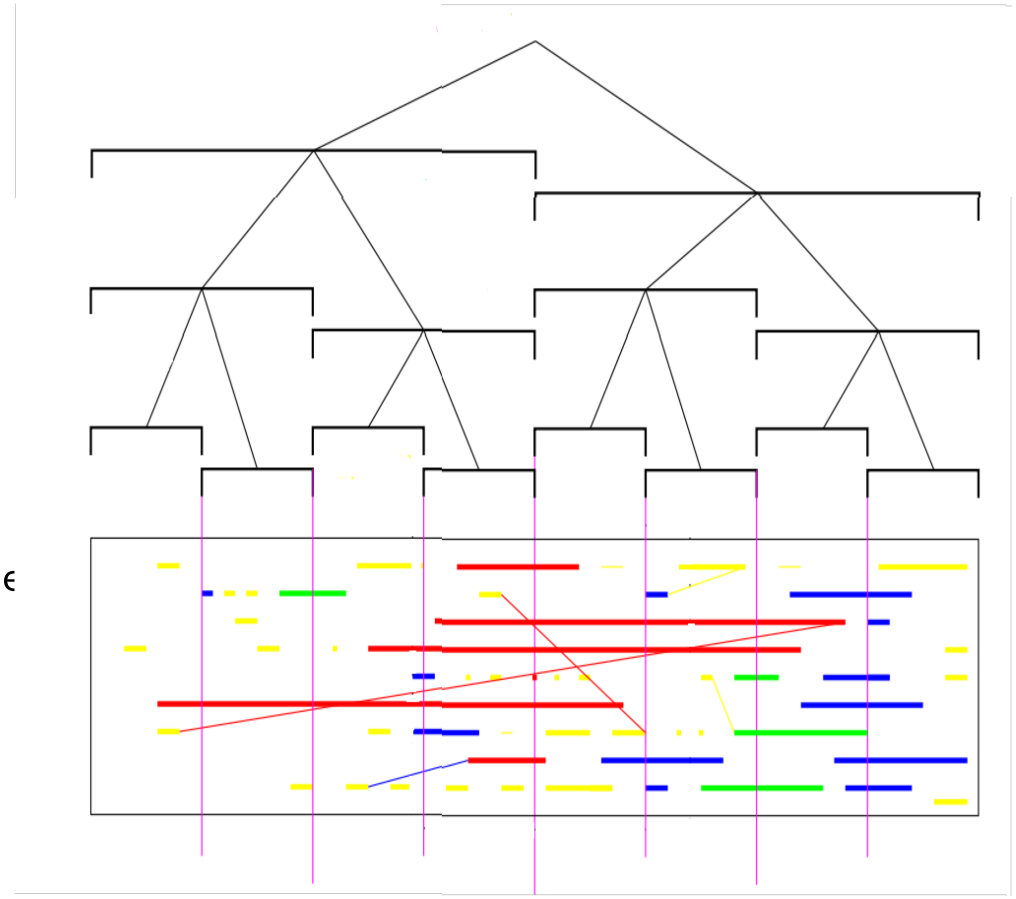
- Compactness: the space efficiency of the data structure.
- Efficiency: the performance of query algorithms.
- Scalability: the possibility of supporting large traces.
- Flexibility: the possibility of supporting different types of hierarchy.

2 How to link the different levels?

Data Structure:



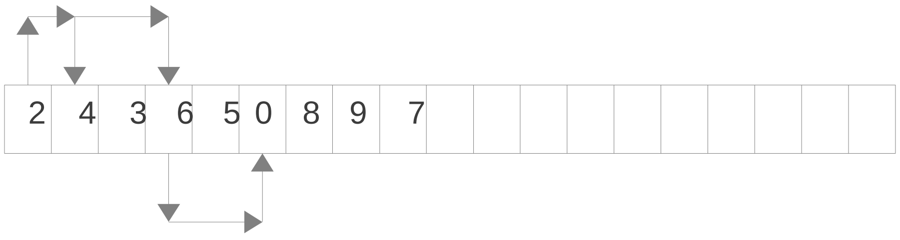
How to link the different levels?



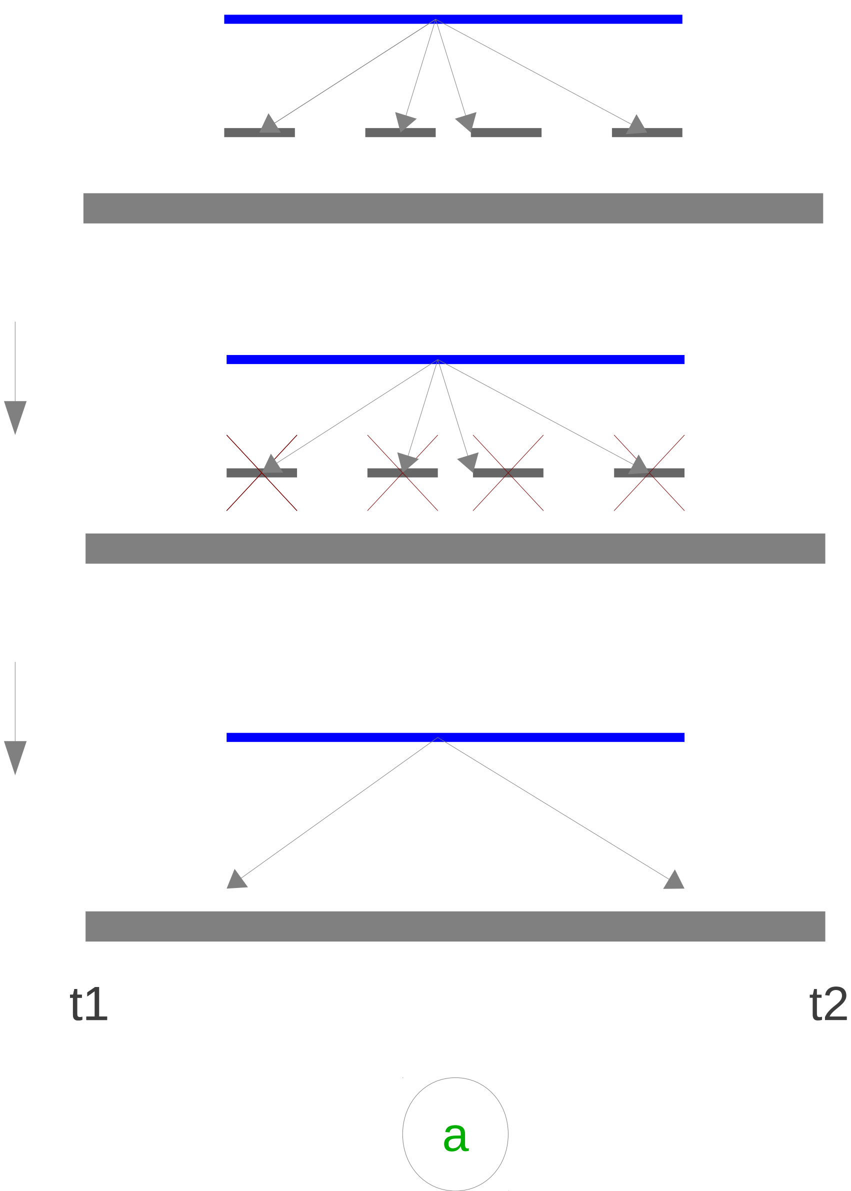
Each item consists of: (29)

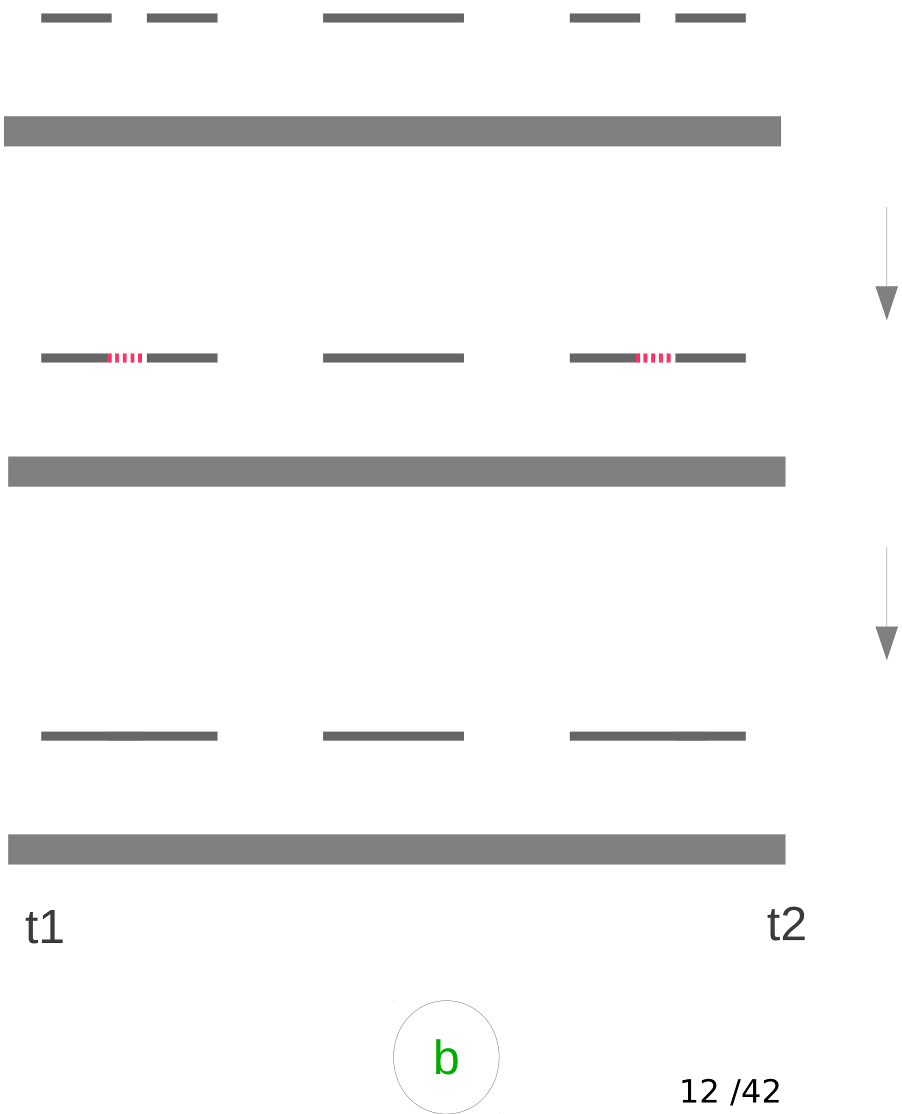
- Level (1)
- Start ptr (6/8)
- End ptr (6/8)
- Number of children (2)
- Address of the first event in the LOD index (4/5)
    - An address is a combination of block/offset address in the index file
    - With a 5 bytes address field, size limit of the index file will be 1 TB. That is relatively large
- Pointer to pattern (2)
- Attribute No (4)

- The “Start ptr”, “End ptr” and the “Attribute No” fields form the MBR (minimum bounding rectangle) of each object.

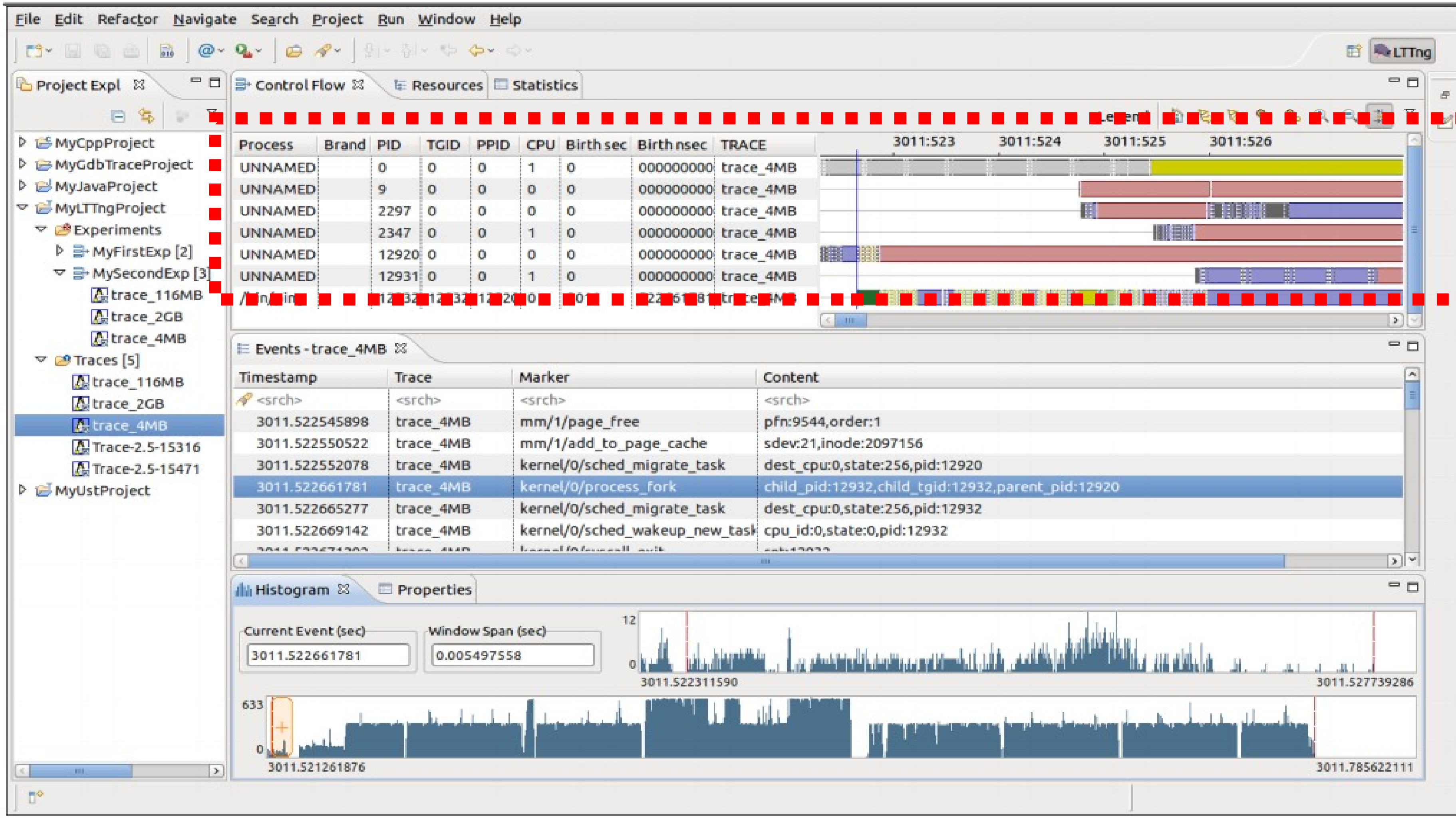


2 How to link the different levels?

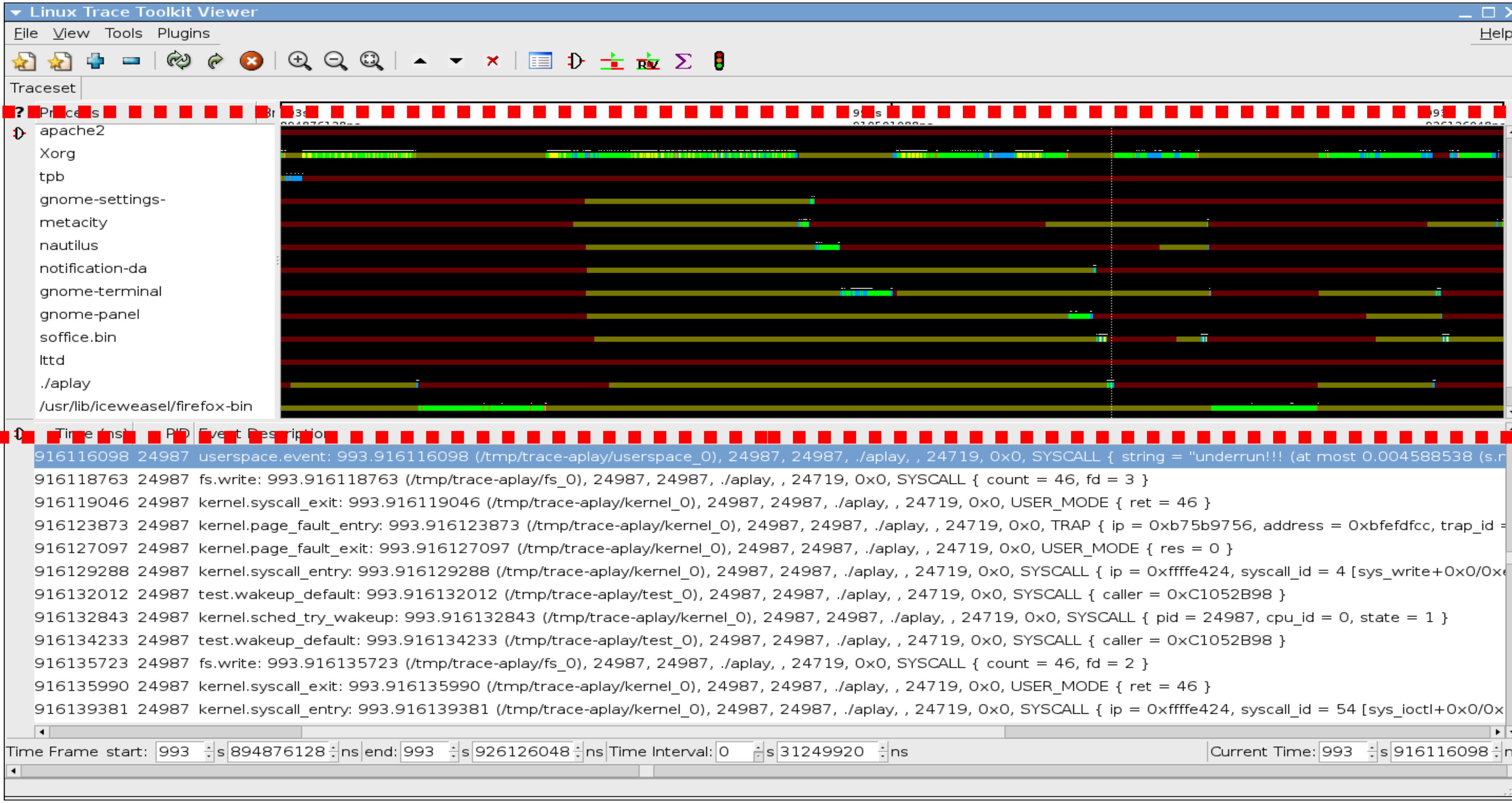




3 How to visualize?



Multiple Coordinated Views

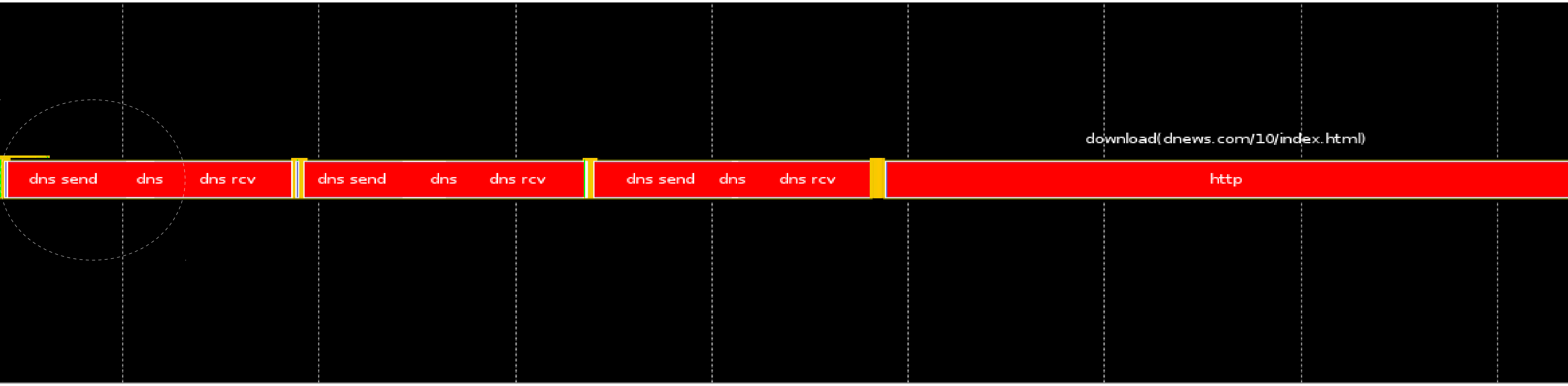


Control Flow Diagram is currently used in the TMF , LTTV and other tools for representing:

- The different stats of a process.
- The function call sequences

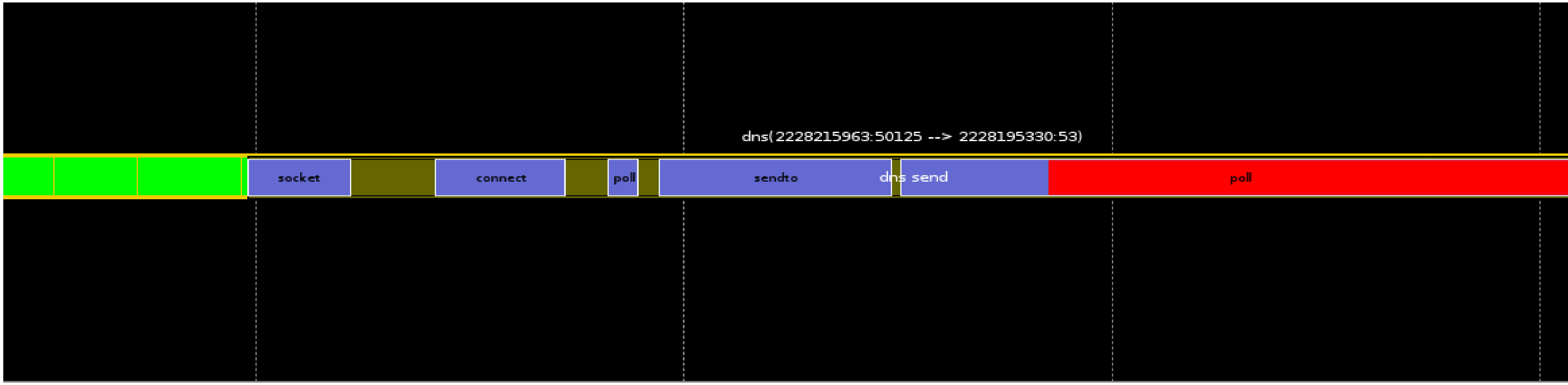
LOD-based Control Flow Diagram

LOD-based Control Flow Diagram



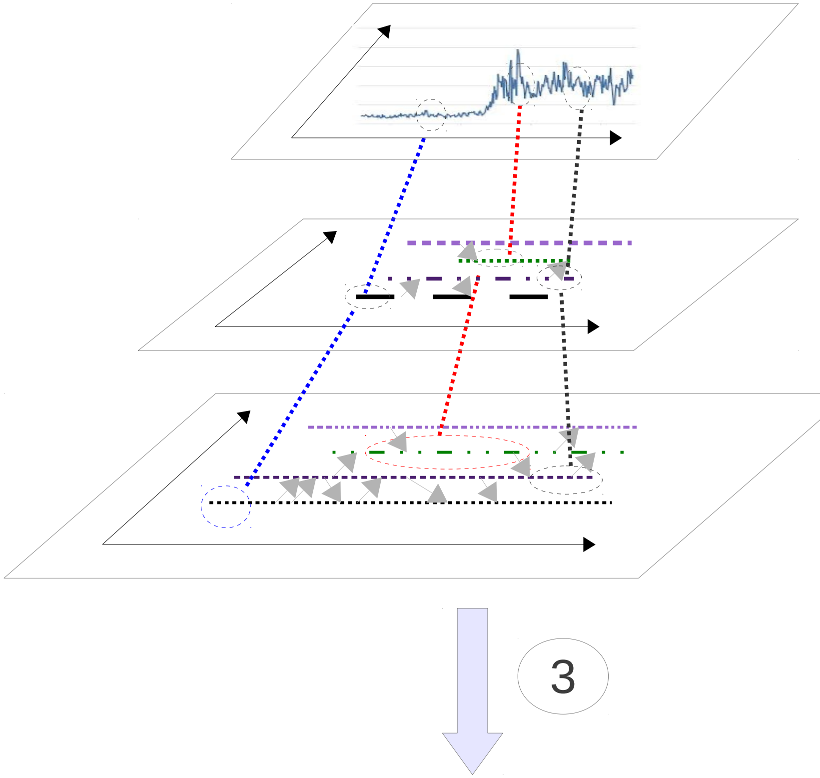
LOD-based Control Flow Diagram

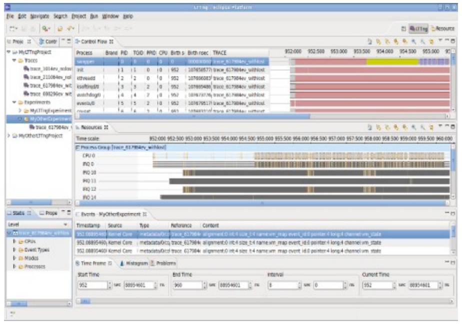
LOD-based Control Flow Diagram



Challenges:

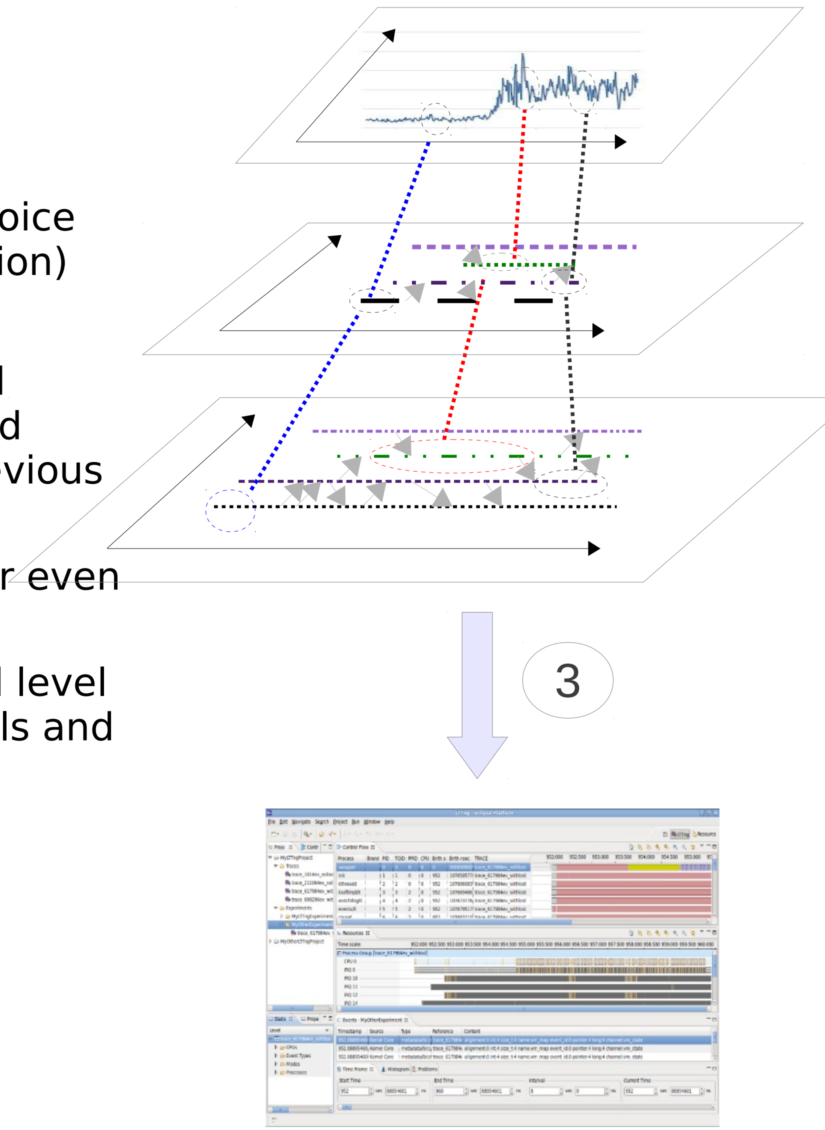
- Drawing the objects in the right level
- Auto label placement
- Aggregation
  - Static (Abstraction)
  - Dynamic (Visual Aggregation)
    - Resource aggregation
    - Event aggregation
- Filtering
  - Size of the event
  - Priority
- Overlapping objects





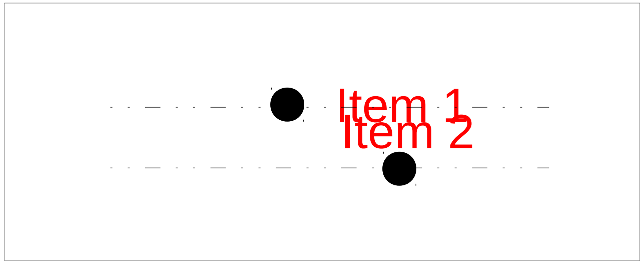
Typical visualization steps:

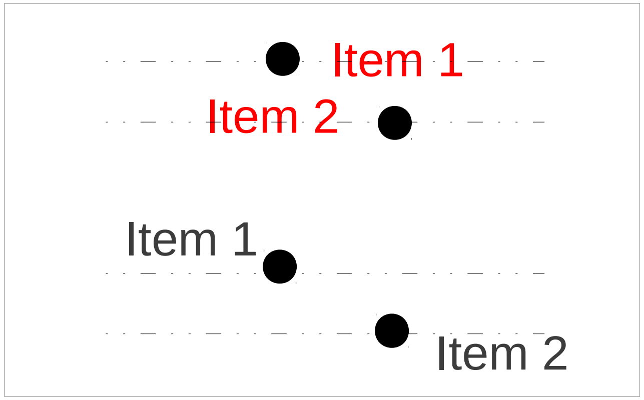
- User selects an area of interest. The algorithm displays the selected window, with a suitable choice of the data level and labels. (high level information)
  - Known scales in the online map system
- User may zoom in to get more data and detailed information. The algorithm displays the area, and presents the labels that were invisible at the previous scale.
- User may zoom for further detail (kernel level) or even the neighbouring areas.
- In the lowest level, the algorithm displays kernel level data (e.g. page faults, interrupts and system calls and so on).

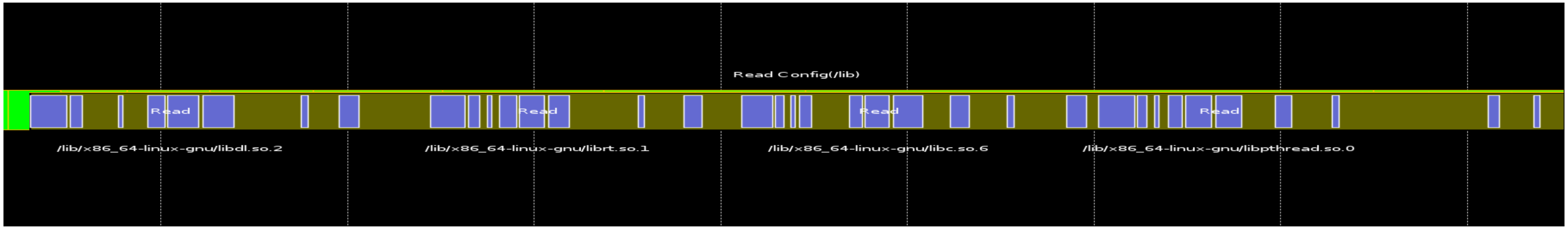


Auto label placement

- Legibly
- No overlap!
  - Other items & other labels
- Clear association of labels with their items
- Fast Algorithm!

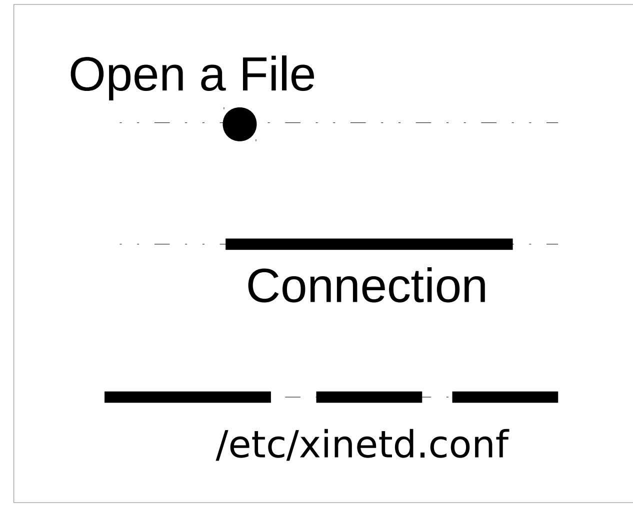


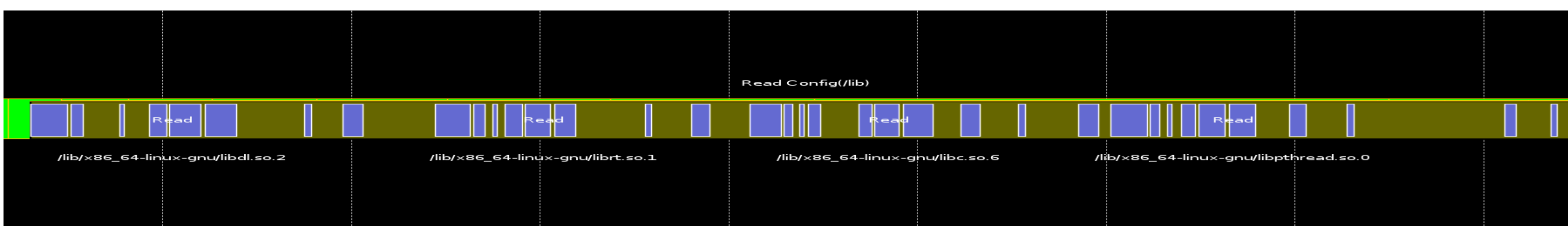




Three types:

- Point labeling
  - Single events, very small states.
- Line labeling
  - States, compound events.
- Area labeling
  - Group of points and lines.

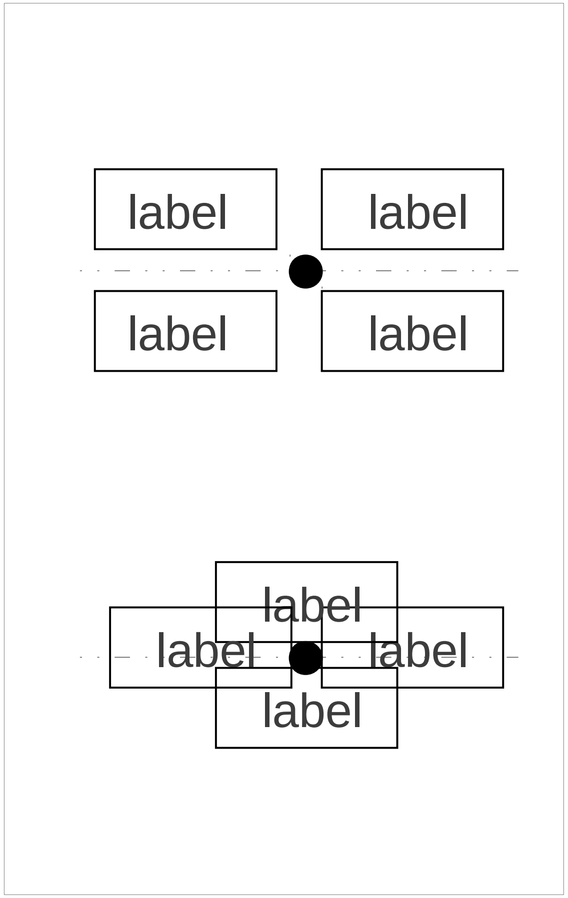


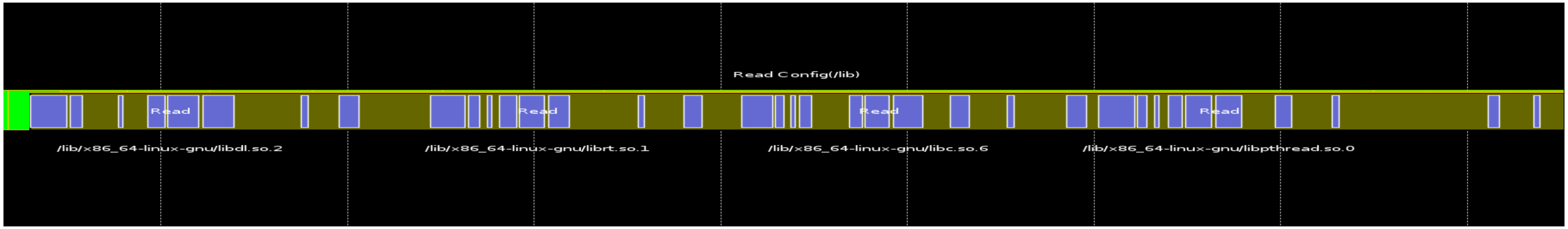


General strategy

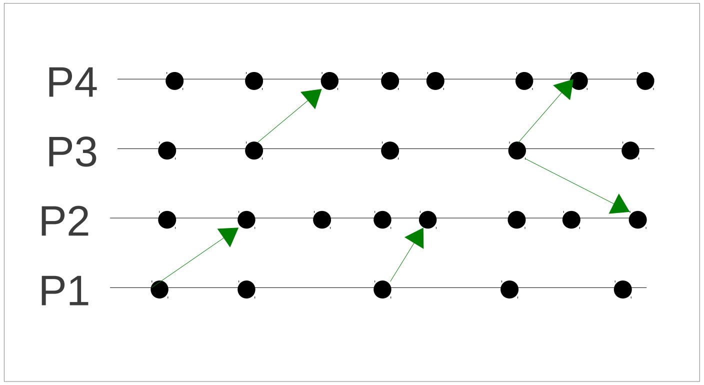
Since the size of the visible window and the scale of data is not known in advance, it is not possible to compute the right positions in advance!

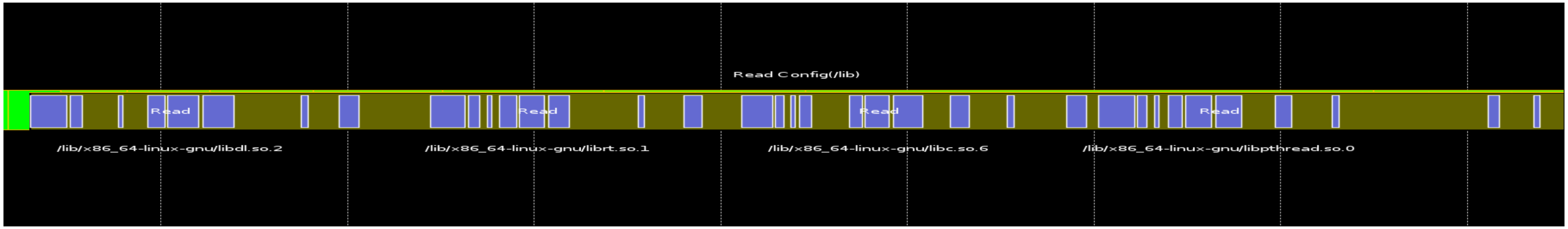
- Generic algorithm:
  - Compute various possible *positions* for each item (rule-based)
  - Select one position for each item
    - No overlap
- It may be impossible to label all items (too small or less-important)
  - Dynamic or static aggregation



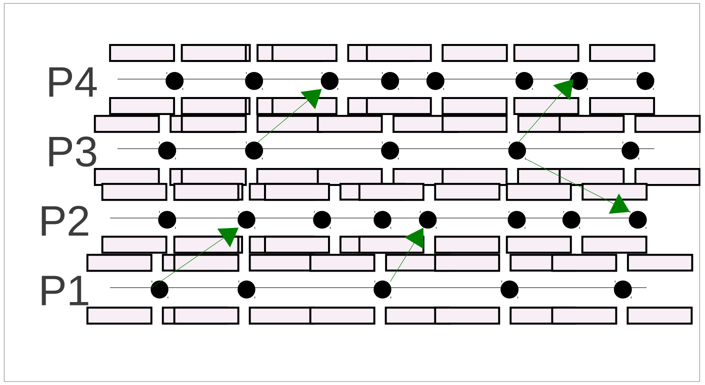


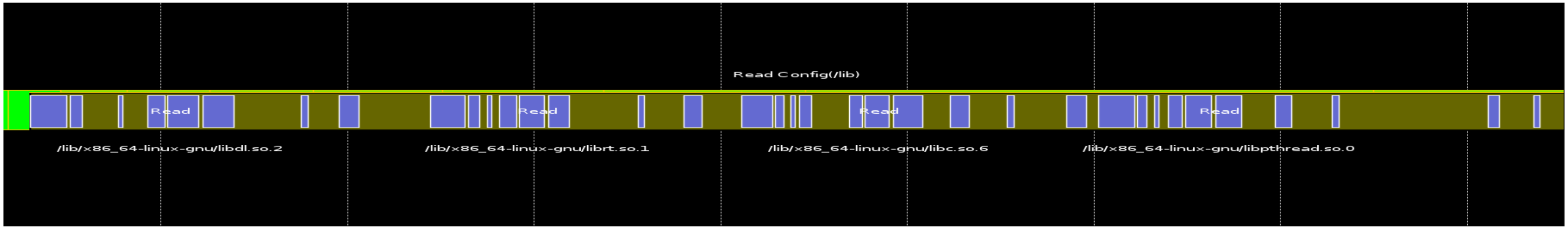
- Using graph theory
  - Extract the items of the visible window
  - Find the label positions for events of each process
  - Draw the conflict graph

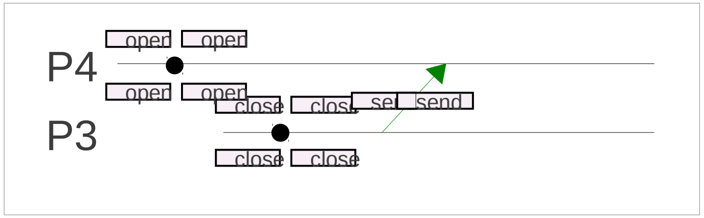


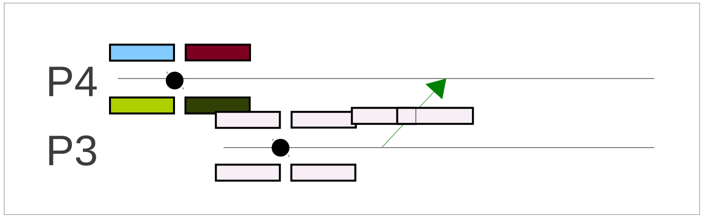


- Using graph theory
  - Extract the items of the visible window
  - Find the label positions for events of each process
  - Draw the conflict graph

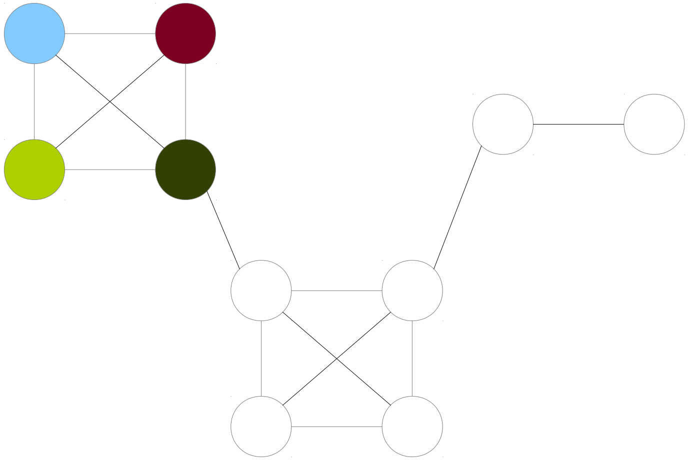


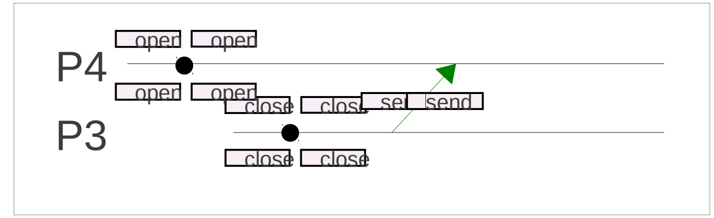




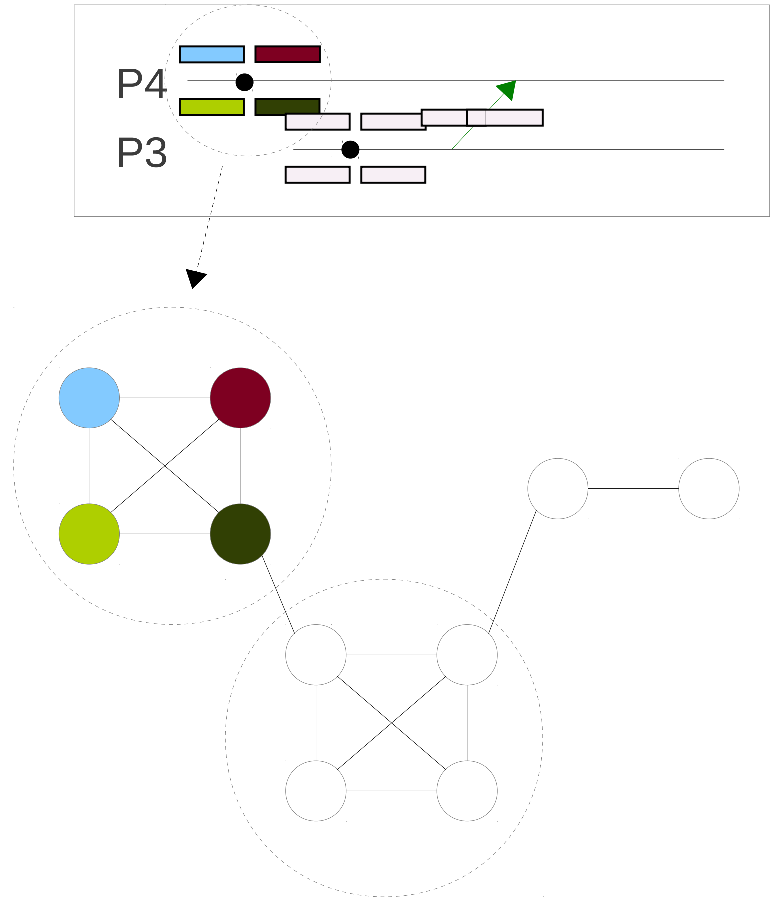


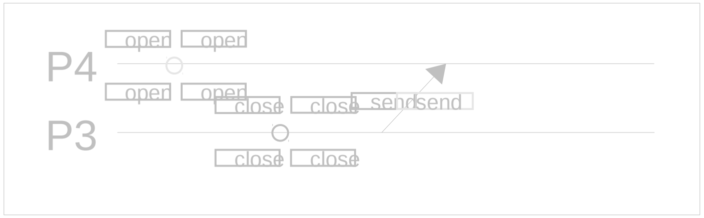
- Using graph theory
  - Extract the items of the visible window
  - Find the label positions for events of each process
  - Draw the conflict graph

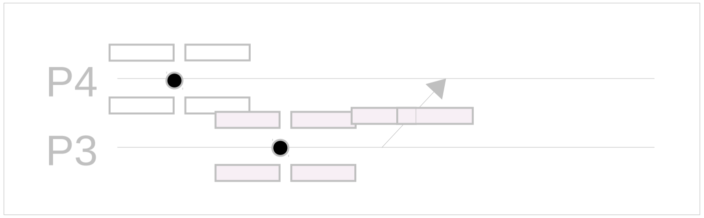




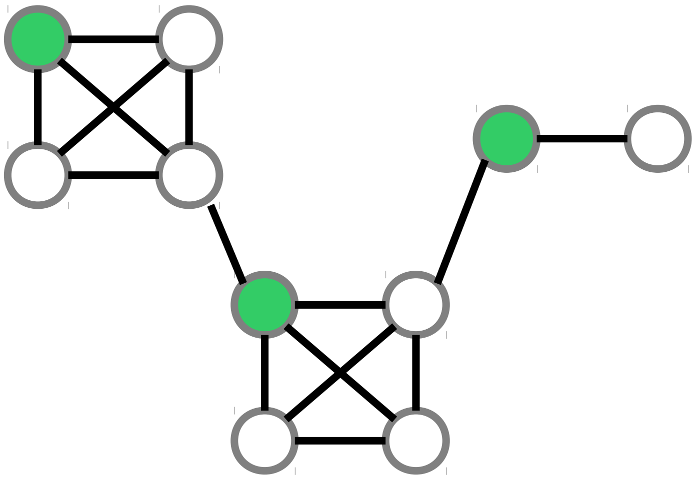
- Conflict graph
  - Each possible label position is a node
  - Each edge shows an intersection in the positions
  - There are edges between each two nodes of the same item.
  - The problem is to find the largest stable (independent) set







- Conflict graph
  - Each possible label position is a node
  - Each edge shows an intersection in the positions
  - There are edges between each two nodes of the same item.
  - **The label placement is equal to find the "maximum stable (independent) set"**
    - **NP_hard**
    - **We try to find a good solution (maximal), instead of the best one!**



Stable (Independent) Set

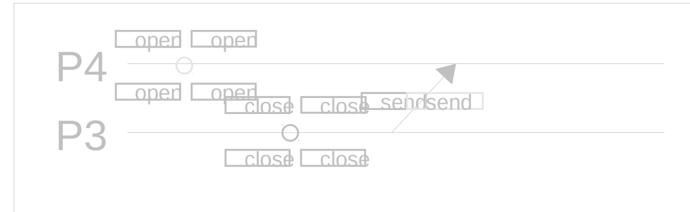
is a subset of nodes, such that no two of nodes are adjacent.

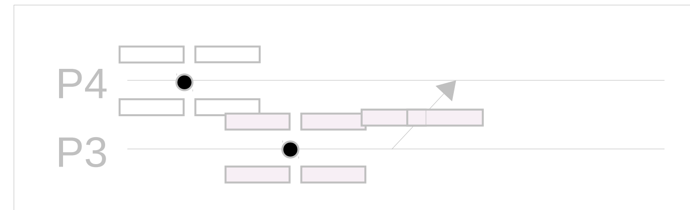
Maximal Stable (Independent) Set

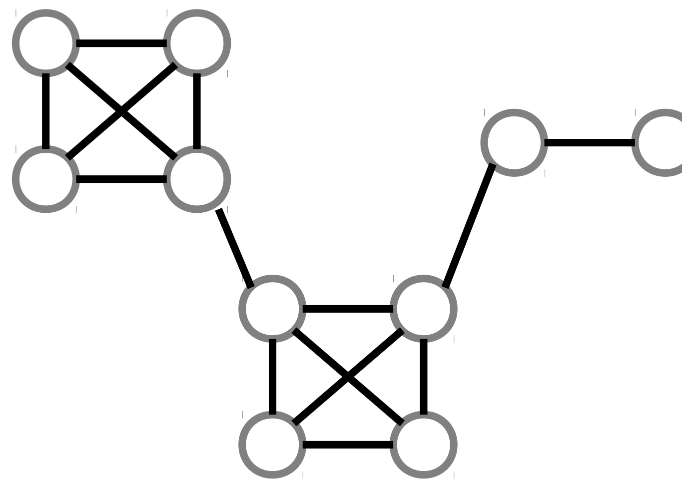
is an independent set that is not a subset of any other independent set. A largest maximal independent set is called a maximum independent set.

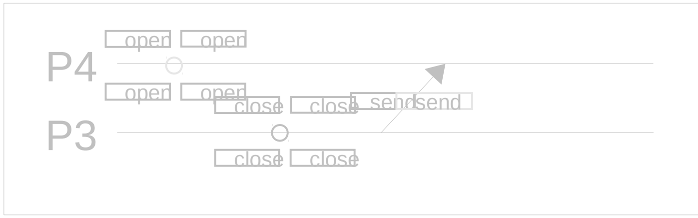
Maximal stable (independent) set

1. S = {}
2. for each node n{
3. if (n has no neighbor in S)
4. add n to S
5. }



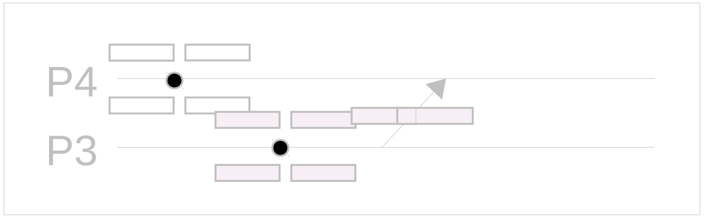


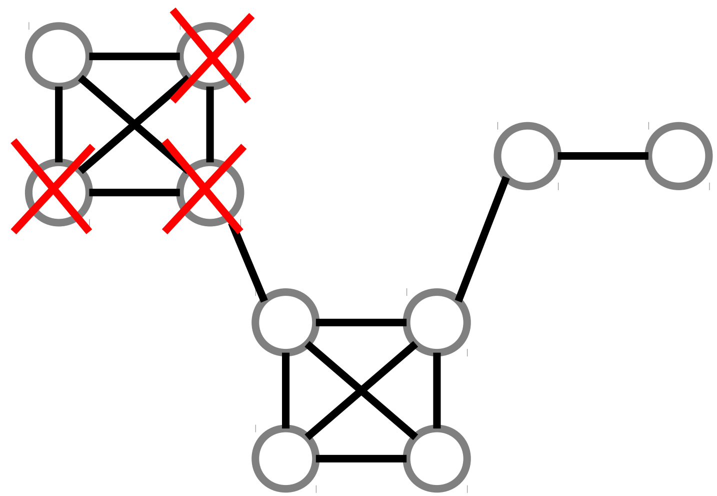




Maximal stable (independent) set

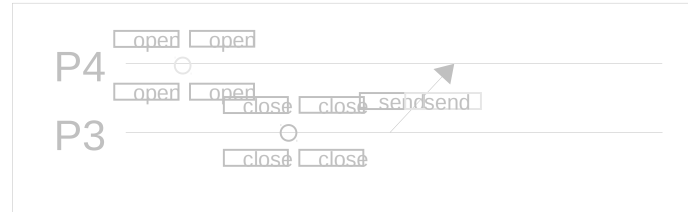
1. S = {}
2. for each node n{
3. if (n has no neighbor in S)
4. add n to S
5. }

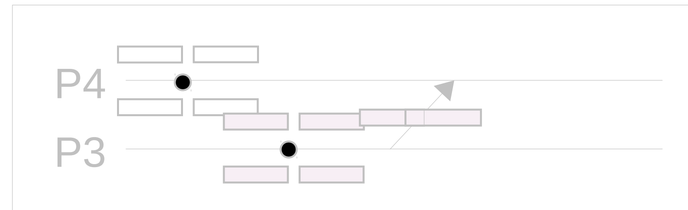


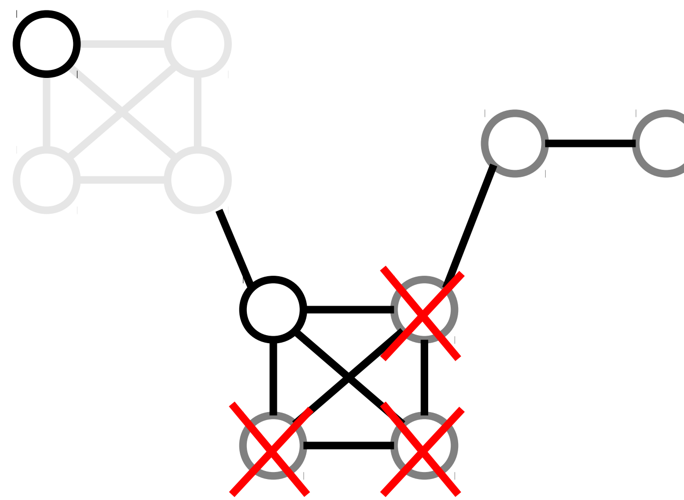


Maximal stable (independent) set

1. S = {}
2. for each node n{
3.   if (n has no neighbor in S)
4.   add n to S
5. }

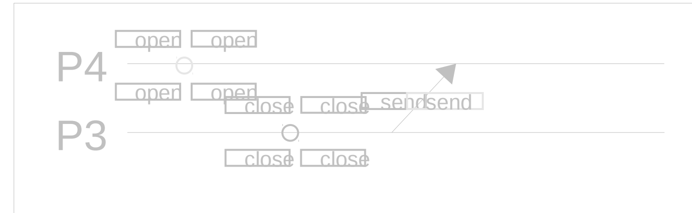


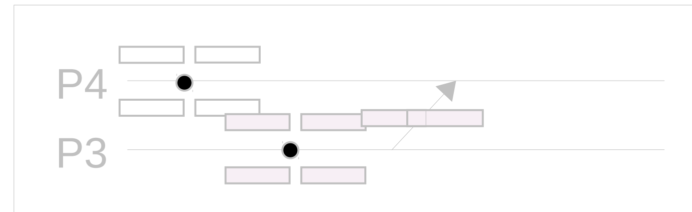


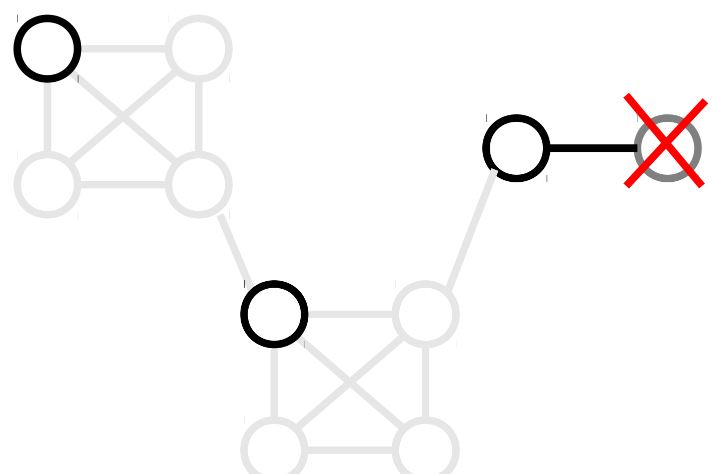


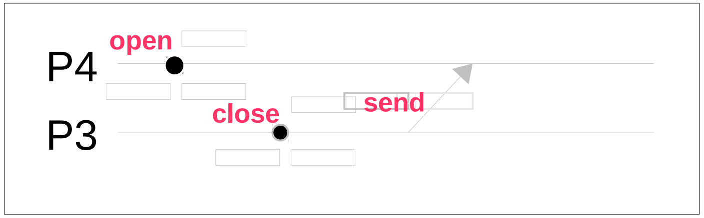
Maximal stable (independent) set

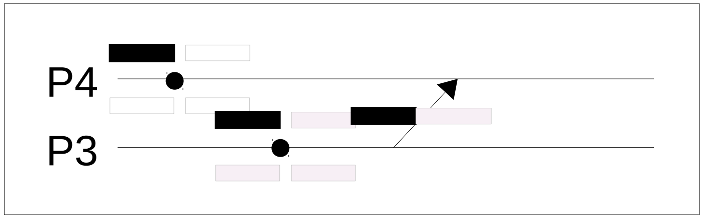
1. S = {}
2. for each node n{
3. if (n has no neighbor in S)
4. add n to S
5. }





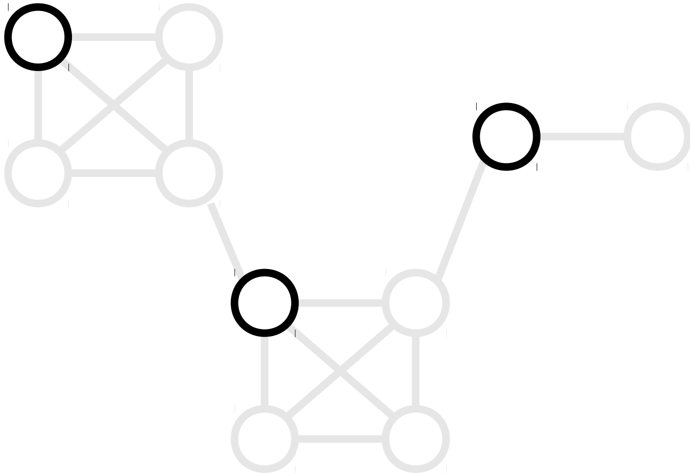






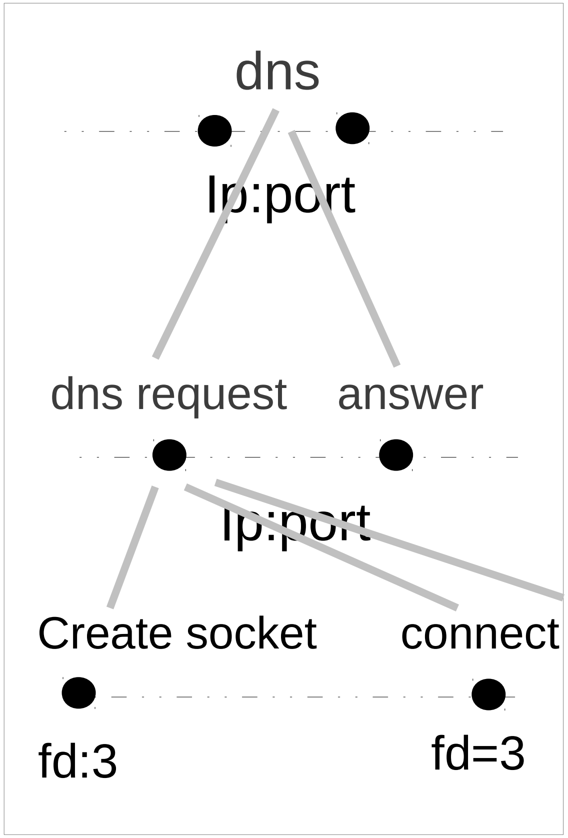
Maximal stable (independent) set

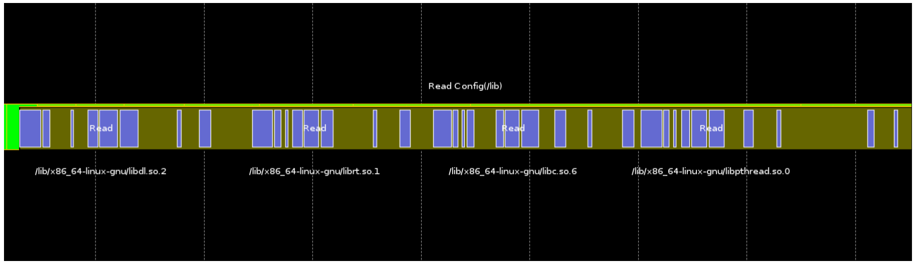
1. S = {}
2. for each node n{
3.   if (n has no neighbor in S)
4.   add n to S
5. }

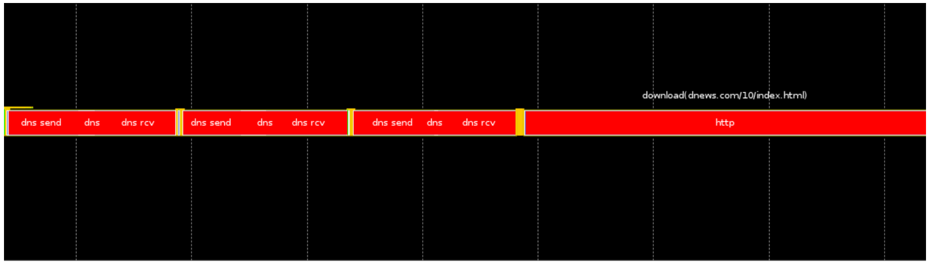


Aggregation

- Select the proper level to show in the view
- **Static aggregation**
  - The annotation data structure that is created in the abstraction and linking phase
  - The size of the visible window and the scale of data is not known in advance.
- Dynamic aggregation
  - Try to dynamically aggregate the items and resources

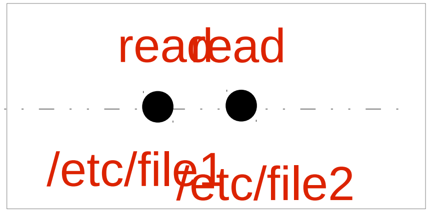


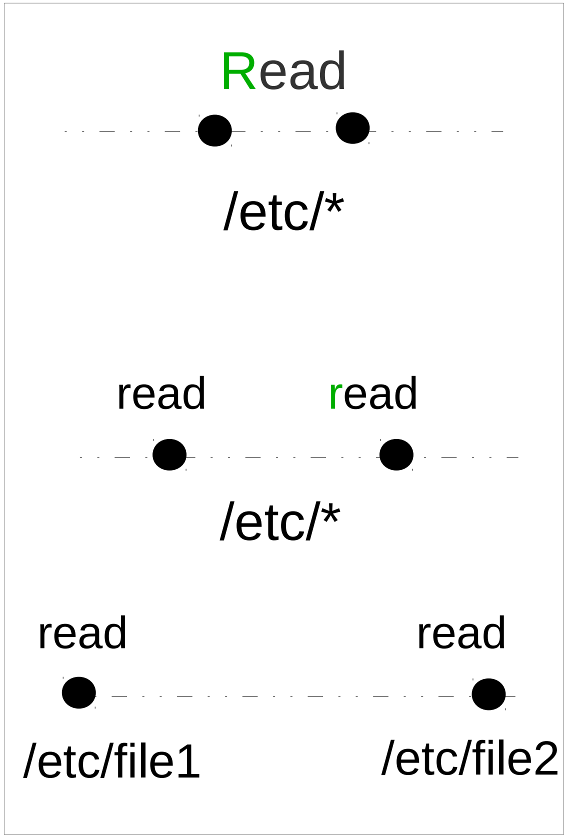




Dynamic Aggregation

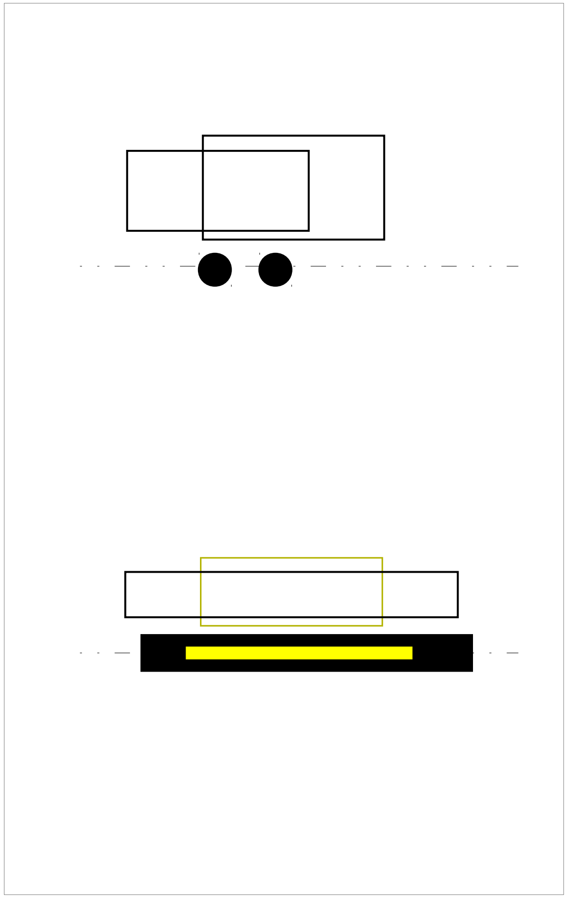
- When zooming out:
  - Remove labels of less important items
  - Put one label instead of a group of the same items and resources.
    - Replace with one aggregated label:
      - Replace folder name for all files of that folder
- When zooming in:
  - Remove labels of items that go outside the visible window
  - Allow more labels
  - Replace (or show together) the aggregated items with their children

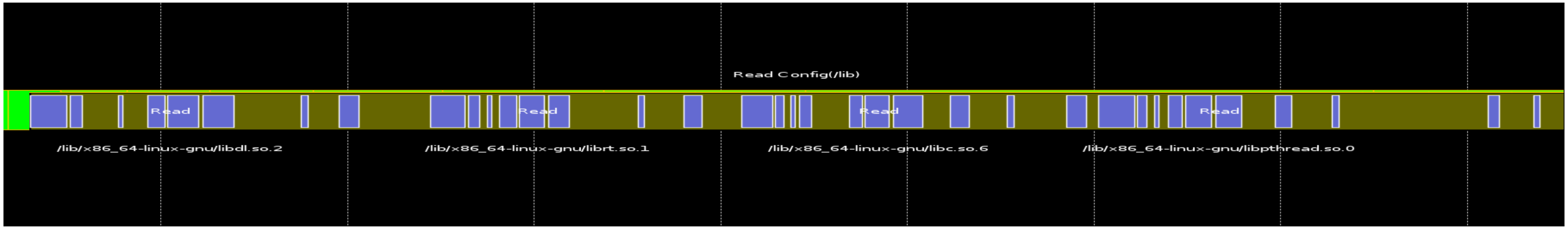




Conflicts

- Conflict between labels
  - e.g. Two adjacent events
  - Label placement algorithm
  - Dynamic aggregation
- Conflict between Operations
  - e.g. Two overlapping “read file” abstract events.
    - A process works with two files simultaneously.
  - Label placement algorithm
  - Dynamic aggregation
  - Displacement
    - Sort based on their start or end times

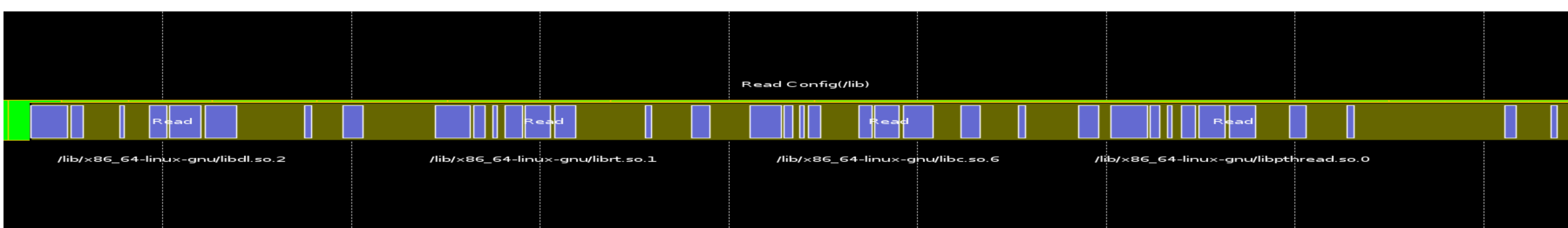




Conclusion

LoD-based Control Flow Diagram

- Firstly shows an overview of the execution
  - The highest level of the hierarchy
- Supports “Level of Details”
  - In this diagram, users can zoom in and focus to get more details
    - Or zoom out to get an aggregated view
- Uses two kind of abstraction methods:
  - Event & data abstraction (static aggregation)
  - Visual abstraction (dynamic aggregation)
- Uses a link data structure to relate the different layers of information together
- Uses label placement algorithm to place name of the items and resources to make it easy to understand what is going on the system execution



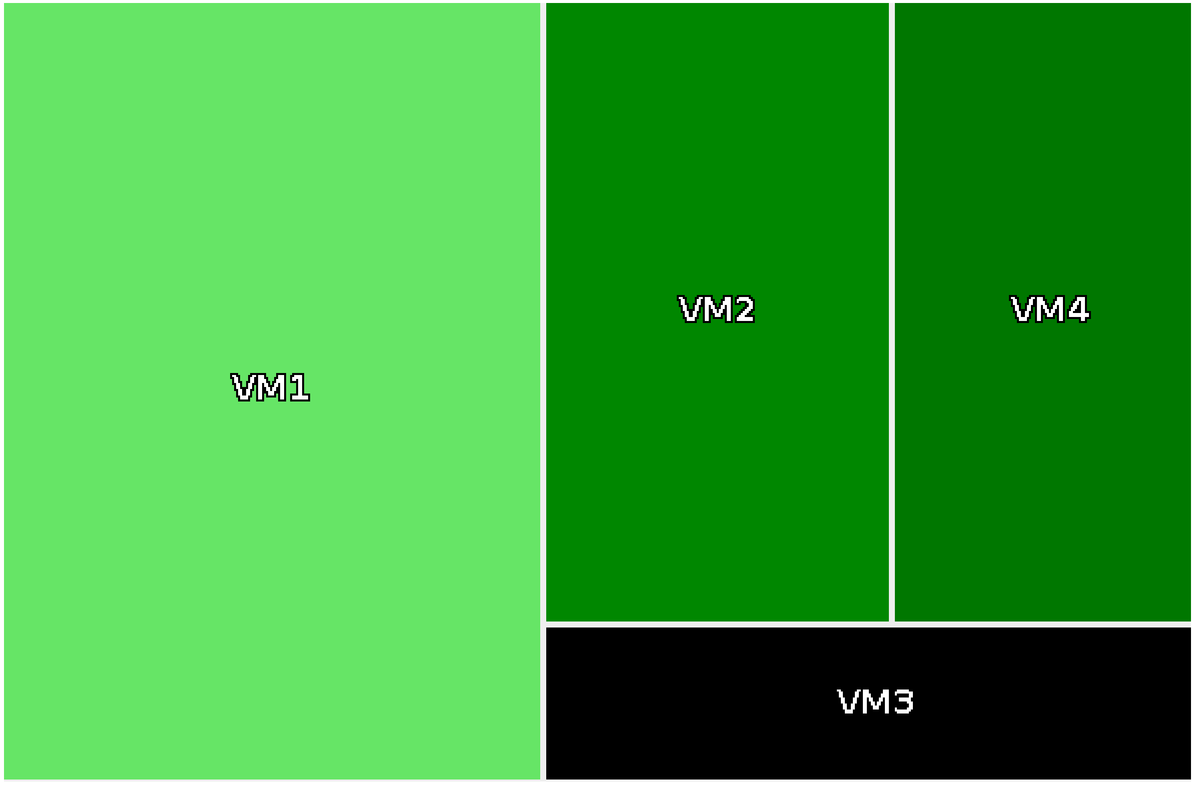
Node-link + Treemap

Other method to visualize hierarchical data.

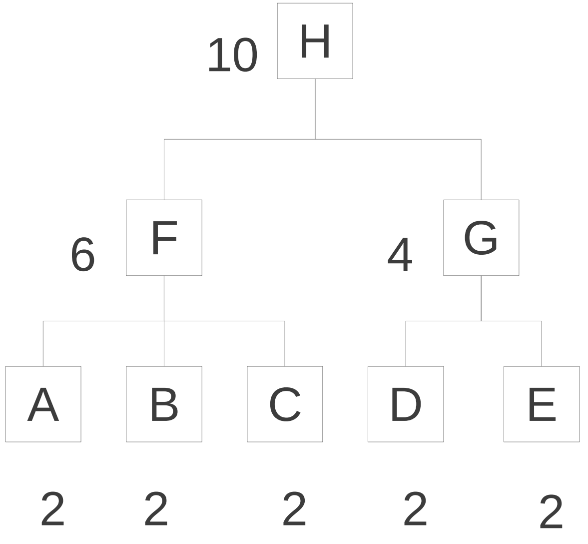
Example: to view data and usage statistics from every virtual machines in the system, broken down in the same way?

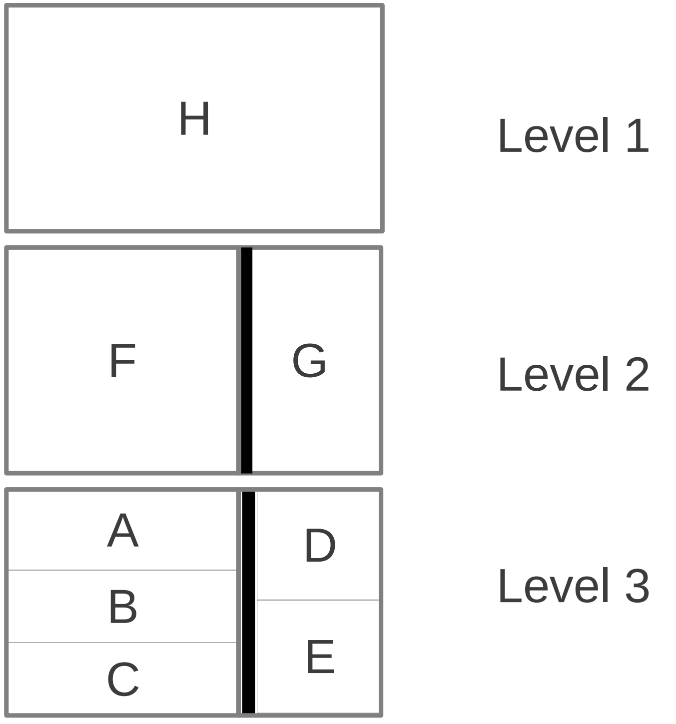
- CPU usage of each VM
    - CPU usage of each process in the VM,
        - each thread of that process,
        - Each CPU for the selected thread
        - ...

- Having a page showing hundreds of pie charts?
- How would these pie charts relate to each other?



- Represent a complex, hierarchical data as a set of nested rectangles and in a relational view.
- The size of each rectangle is based on some measure.
  - Another measure, (e.g. time since the last modification to a file) can be used to color each rectangle.





Demo

Thank you

n.ezzati@polymtl.ca